

11

200
<HTML>
<HEAD>
<TITLE>00 Objects/Classes/Instances</TITLE> 220
</HEAD>
<BODY>

<P>
 Understanding Object Orientation Concepts 232
</P>

<P>
 Objects & Classes 234
</P>

<P>The world is full of <I>objects</I>. We naturally think of objects in hierarchical categories, or <I>classes</I>. For example, a computer is a general class of object. A hierarchy of object classes surrounds the class "computer", extending in both directions. "Computer" is a member of the more general class "machines". In the other direction of the hierarchy are specific types of computers: notebook computers, supercomputers, HP computers, etc. If you are reading this document on your computer, you are looking at an <I>instance</I> of the class "computer".</P> 236

<P><CENTER>
 240
</CENTER></P>
<HR /> 250

Mo'00? Look up another concept:
 238

260 {<TABLE border="2" width="60%">
 <TR>
 <TD>
 Inheritance
 <TD>
 Encapsulation 270
 <TD>
 Overloading
 </TR>
</TABLE>

</BODY>
</HTML>

Fig. 2

```

main ()
{
310   {
      htmlDocument* document = new htmlDocument (stdout,
                                                 "00 Objects/Classes/Instances");
      tableGrid*      table      = new tableGrid (1, 0, 0, "60%");
      centered*       center     = new centered ();

      document->add( new paragraph () );
332   {
      document->add( new htmlText ("Understanding Object Orientation
                                    Concepts", "forestgreen", normal, bold, "+2", "arial"));

      document->add( new paragraph () );
334   {
      document->add( new htmlText ("Objects & Classes", "black",
                                    normal, bold));

      explanation = query(oo_concept_database, concept);
      find_first_and_italicise(explanation_text, "object", "class",
                               "instance");
      document->add( new paragraph() );
      document->add( new htmlText(explanation_text) );

      document->add( new paragraph());
340   {
      center->add( new image(explanation_image) );
      document->add(center);

      document->add( new horizontalRule() );
350   }

338   {
      document->add( new htmlText("Mo' OO? Look up another link:") );

      table->newRow();
      table->addField( new anchor("http://www.mooo.com/Inheritance",
                                   new htmlText ("Inheritance") ) );
360   {
      table->addField( new anchor("http://www.mooo.com/Encapsulation",
                                   new htmlText ("Encapsulation") ) );
      table->addField( new anchor("http://www.mooo.com/Overloading",
                                   new htmlText ("Overloading") ) );
      document->add(table);

      delete document;
    }
}

```

Fig. 3

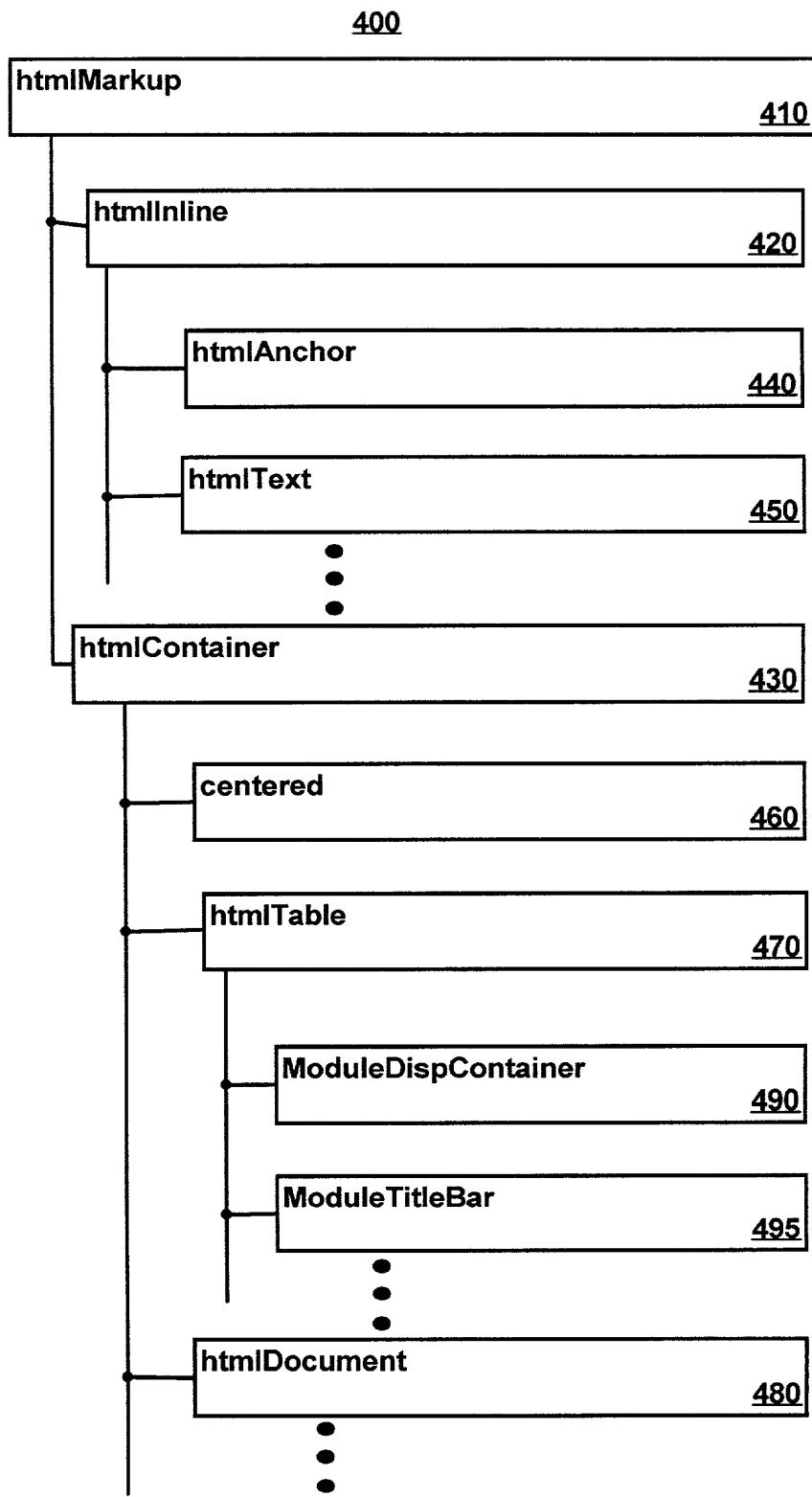


Fig. 4

410

```
// This class is an interface for defining the basic HTML/XML
// relationship between a child element and its parent.

class htmlMarkup
{
protected:
    htmlMarkup* parent = NULL;
    FILE* fptr = NULL;
public:
    htmlMarkup();
    virtual ~htmlMarkup();
    virtual setParent(htmlMarkup* parent) { this.parent = parent } 510
}
}
```

Fig. 5

420

```
class htmlInline extends htmlMarkup
{
protected:
    DynamicArray* buffer = NULL;
public:
    htmlInline();
    virtual ~htmlInline()
        { if (buffer) fprintf(parent.fptr, "%s", buffer) } 610
}
}
```

Fig. 6

440

```
class htmlAnchor extends htmlInline
{
public:
    htmlAnchor (String href, htmlMarkup* label) { 710
        { buffer = "<a";
          buffer += " href=" + href;
          buffer += ">";
          // flush the label markup to this buffer
          label.setParent(this);
          delete label;
          buffer += "</a>";
        }
}
}
```

Fig. 7

430

```
class htmlContainer extends htmlMarkup
{
protected:
    FILE*fptr = NULL;
public:
    htmlContainer();
    virtual ~htmlContainer()
    { if (fptr && parent.fptr)
        concatenateFiles(fptr, parent.fptr);}
}
```

Fig. 8

450

```
class htmlTable extends htmlContainer
{
public:
    htmlTable()
    { fptr = new temporaryFile();
        print("<table>");
    }

    virtual ~htmlTable ()
    { print("</table>"); }

    void addRow()
    { print("<tr>"); }

    void addContent(htmlMarkup* content)
    { print("<td>");

        // flush the child content to this table
        content.setParent(this);
        delete content;

        print("</td>"); }

}
```

Fig. 9

1000

| Class | Style | HTML element |
|------------------|---------------|--------------|
| commentText | htmlInline | <!-- -- > |
| htmlText | htmlInline | ASCII text |
| formattedText | htmlInline | <PRE> |
| embeddedText | htmlInline | <LAYER> |
| htmlImage | htmlInline | |
| htmlAnchor | htmlInline | <A> |
| paragraph | htmlInline | <P> |
| centered | htmlContainer | <CENTER> |
| lineBreak | htmlInline | |
| noLineBreak | htmlInline | <NOBR> |
| horizontalRule | htmlInline | <HR> |
| table | htmlContainer | <TABLE> |
| htmlDocument | htmlContainer | <HTML> |
| htmlForm | htmlContainer | <FORM> |
| formInput | htmlInline | <INPUT> |
| formTextReadOnly | htmlInline | <TEXT> |
| selectionList | htmlContainer | <SELECTION> |

Fig. 10

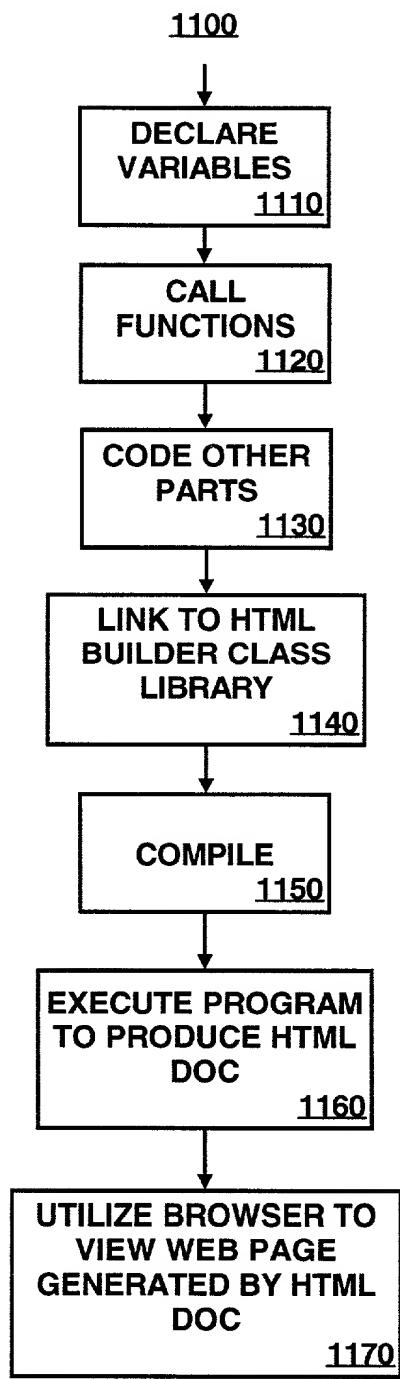


Fig. 11